CHAPTER 22

Packaging and Deploying Windows NT Applications

In This Chapter

Understanding the various packaging models and where they apply

Learning which packaging model to use for specific situations

Creating a package that has everything the user needs

Using the InstallShield packaging program

Advertising product features during an application install

We've spent a lot of time looking at the various kinds of Windows 2000 applications that you can create. Obviously, this is the "main event" for most programmers because it's what we're trained to work with. However, there's another element in programming that all too often gets neglected simply because no one takes the time to really learn how to do it—packaging.

The packaging that an application comes in is more important than many people think, because it determines how others view your application. That first impression is important and it's the package that gives users their first impression of your product, not the application itself. In fact, packaging is so important that it's one of the items that all software reviewers include on their list.

Obviously, the first thing that we need to discuss is what I mean by packaging. The one thing I'm not talking about is the box that the application and manuals come in when you see them at the store. Most of you will never even work on that kind of application. The packaging I'm talking about is the installation program, help files, README file, and other add-ons that don't affect the way the application works, but do affect the user's impression of your application.

Of all the packaging elements, the installation program is the most important. Most users will simply stick the CD for your application in the drive and expect you, the programmer,

to lead them by the hand from that point on. How well your installation program does in getting the application installed will definitely affect how the user views your application. In fact, a bad first impression during installation can flavor the user's perspective of your application from the outset and ruin what might be an otherwise good experience.

Creating a package that works well with your target audience is critical. The first two sections of the chapter addresses the issue of what kind of package you should create given a specific kind of usage environment and target audience. Some users want continual help; other users want you to tell the basics and not insult their intelligence by providing help with relatively easy tasks. Of course, there are users who fall into another category and want an installation that requires no work at all on their part. In addition, there are installation program elements that you'd include with a commercial application but that you don't need to include with an application created for enterprise (corporate) use.

Once you know what kind of package you want to create, you need to decide what to put in the package. The next section provides insights into the entire packaging process. It helps you understand what the user will need to consider a package complete and easy to use.

InstallShield is the Microsoft-supplied application used to create installation programs for Visual C++. For most developers, InstallShield is all they'll ever need to create a complete installation program, and it provides everything the user needs. Just look at the number of commercial applications that use InstallShield as the basis for creating their installation programs. Though the version of InstallShield that comes with Visual C++ isn't the complete product, it does do the job well enough that even enterprise (corporate) developers will have everything needed to create a good package.

We'll be using InstallShield for Visual C++ 6 to create the installation program in this chapter. This product is actually a subset of InstallShield Professional Version 5.5. You can find out about InstallShield 5.5 and a lot of other useful packaging products at `http://www.installshield.com/`. This site allows you to download a trial version of either InstallShield Express (which is similar to what you got with Visual C++) or InstallShield Professional. Check out `http://www.installshield.com/ispro/vc6.asp` if you want to get a full comparison of the Visual C++ version versus the Professional version. You can also call (800) 374-4353 or (847) 240-9111 to find out about the trial version and obtain a copy of it.

Almost no one reads the README file, and of those who do, most won't readily remember the contents of the README file for more than a few minutes. One way that large companies, such as Microsoft, make the README file more user-friendly, is to create advertisements. The advertisements get displayed as the user installs the program and makes them aware of essential product features and new developments. The next section of the chapter shows how you can add these advertisements to your next installation program so that you can be fairly certain the user will at least know the essential features of your product.

An Overview of the Packaging Models

There are a lot of different considerations when it comes to creating a package to deliver your application in. The most important consideration for most developers is the environment you plan to use the application in. An in-house developer won't have the same concerns as a software company who plans to deliver an application for use by an outside party. For one thing, the in-house developer probably won't have much time to get an installation program together — time becomes a factor in most in-house situations.

However, there are some common considerations, no matter what kind of application you want to create. For example, it really doesn't matter who will perform the installation or what kind of installation they plan to perform, you still need to write some type of installation program in most cases. The first section in this topic will address this and other common concerns. Make sure you place these common concerns on your to do list, even if you're only a company of one.

The next four sections look at the four most common types of installation environments in this section of the chapter: small-to-medium-sized company, enterprise, shareware, and shrink-wrap. You'll likely fall into one of the four categories, but may want to look at the other categories as well because they contain a lot of helpful tips and hints. Some of you won't fall into a distinct category. For example, a consultant is part in-house (corporate) developer because most of the applications the consultant creates are customized for a particular company. On the other hand, your client will want a little more out of a consultant than something that works. The client will want something around the level of a shareware package as a minimum (some clients expect a shrink-wrap version of your product, but that's a little on the unreasonable side).

Suffice it to say, figuring out what kind of packaging to provide is tricky, but not impossible. All you really need to do is assess the needs of the people you're writing the application for. If all they're looking for is something that works well and doesn't include a lot of hype, then using the enterprise (corporate) packaging style will work just fine.

As more and more applications require the space offered by CD-ROMs and DVDs, it's becomes important to know how to create a bootable version of your disk. Creating a bootable CD-ROM isn't impossible. In fact, there's actually more than one way to do so (Microsoft actually uses two different methods on Windows 95 and Windows NT). One of the better ways to create a bootable CD is found at the CD-ROM Professional site: `http://www.onlineinc.com/cdrompro/CP1995/AugCP95/starrett.html`. This particular technique relies on a new feature found in many BIOS chips that allows for CD-ROM access prior to loading the operating system-specific drivers. In addition, you'll likely find that some newer CD recording software also provides this feature in a more automatic format.

Common Concerns

No matter what kind of application you've written and whom you've written it for, there are some common elements that everyone needs to think about. This section looks at the common concerns that every developer should have, no matter what kind of application package they put together.

Obviously, the first thing you need to put in the package is all of the required files. This may seem obvious, but there have been a number of times when I tried to install an application only to discover that it didn't include everything. Consider, for a moment, the MFC library files. Some developers might not include them with their application, but the user might have an old version of the files that won't work with your application. The end result would be a broken application and a disappointed user. One way to look at this particular need is the "batteries not included syndrome." Always include the batteries along with the new application.

Packaging also has a lot to do with an application's ease of use and the user's ability to install it quickly. Imagine for a moment a car that wasn't packaged correctly. Sure, it comes with directional signals, but there isn't any light on the dash to tell you which of the directional signals is blinking. The engineer knows which way the directional lights work, so why bother to install indicator lights? You wouldn't buy a car like this, and neither would anyone else. That's what an application without a good help file is like to a user who didn't design it. You may as well just tell them to guess about how to use the application because they'll never think to look at the README or other file you placed on the disk with a modicum of instructions.

Everyone should make his or her application look aesthetically pleasing because appearance affects the user's attitude toward the application. Sure, you don't need to provide brilliant graphics worthy of an art gallery for an in-house application, but even a little color will dress up the display. I've found that attitude is about half the battle in getting a user up and running with a program. Anything you can do to improve the user's attitude also benefits you by reducing service calls and complaints. In short, bells and whistles are nice, but a good interface is required. Even your installation program has to have a great interface; after all, it's the first part of your program that the user sees. First impressions are very long lasting—especially if you're a user who didn't want to use an application in the first place.

A complete application always includes an install program that you can use to test the setup as a whole. Here's another scenario to think about. One programmer I know of decided to simply send out a batch file and a disk full of application files as a package. He never even bothered to test the application because it was only going to be used in-house. It didn't take too long before the programmer started getting a rash of calls from disgruntled users. It seems that he forgot to add a crucial file to the disk, and the users who were actually able to figure out the batch file couldn't get the program to work. No one ever liked that application—even after the programmer added a nice-looking interface and an install program, and fixed the bugs. The problem wasn't with the program; it was with the user's attitude toward the program. First impressions are crucial to a program's success.

Enterprise (Corporate)

Enterprise-level programmers have the least to do with the beautification of their packaging and the most to do with customizing it. Let's face it, the enterprise programmer really doesn't have to impress anyone with fancy graphics or impressive sound effects. That the interface is both functional and user-friendly is enough. Frankly, your boss may not look very favorably on any time spent making an installation program look beautiful, especially if you have a stack of other programming projects awaiting your attention.

On the other hand, the enterprise programmer is most often engaged in creating custom applications that reflect a specific company's needs. It's not all that uncommon for custom applications, especially database applications, to require the use of specialized files to store both temporary data and global settings. You'll likely need custom registry entries and a wealth of other custom settings as well. Creating a custom installation program that's guaranteed to work on all the workstations in an enterprise (corporate) environment with little help from the programmer is quite an undertaking. Plan on spending a lot of time hand-tuning the standard installation program you can create with products like InstallShield.

If you've used a file located on the server or a central database to maintain application settings in the past, consider placing these settings in Active Directory instead. Using Active Directory allows you to maintain the settings easier, improve security, and reduce application overhead. In essence, Active Directory replaces global settings files much as the registry replaced local INI files.

Media is another place where most enterprise (corporate) installation programs differ from the other categories. It's still not unusual (although it's becoming more so) for a shareware application to come on floppies. However, shareware vendors are about the only ones using floppies. Even shrink-wrap applications use CDs instead of floppies because of their superior storage potential and higher access speed (when compared to floppies). As an enterprise programmer, you need to find a fast and efficient solution for your installation media needs. You should consider CDs the minimum media for distribution your application.

Fortunately, there's probably a better choice of media at the enterprise developer's disposal. Unless the company you're working for is living in the dark ages, you'll probably have a LAN on hand for distribution purposes. You can use this feature to your benefit when creating an application package. Unfortunately, some installation-development programs don't support LAN installation. In this case, you can simply select the most efficient storage method possible — a CD. You'll find that you can create and test your installation program a lot faster if you rely on a CD version of the package and use a LAN to distribute it.

InstallShield 6 Professional Edition comes with the capability to install an application over an intranet. Companies that occupy more than one building can simply extend the LAN distribution principle we're talking about here to distribute the application as widely as needed. Make sure you spend some time looking at the various distribution methods that InstallShield and other packaging applications provide before making the decision to commit your application to floppy or CD and distribute it manually. Using a LAN or intranet to distribute your application is almost always more efficient in the enterprise (corporate) setting than any other technique.

Once you've packaged your application on the LAN, simply add an instruction to the main logon batch file for your server to install the program. The next time the user logs on, your logon batch file will check to see whether the application is installed. If not, it'll call the installation program that you've placed on the LAN.

The enterprise (corporate) setting is one place where using UNC paths in place of standard drive identifiers is very handy. Using a UNC path for the source directory ensures that everyone is able to access all the required source files for your application with little effort.

There are some other packaging issue suggestions that you may want to consider for the enterprise (corporate) environment, although it's by no means certain that you'll actually be able to use all of them. The following list of ideas is meant simply to provide some things for you to think about when customizing your installation program. Whether they'll work in your particular situations depends on how your enterprise (corporate) structure is set up, as well as on environmental factors such as the availability of network connections for the user.

Centralized Common File Storage

The standard procedure for distributing applications is to place all of the files on the user's machine. When you think about it, this is the only way to do things with shareware or shrink-wrap software. As an enterprise (corporate) programmer, though, you can choose to keep specific common files, such as DLLs, on the file server instead of the user's machine.

There are two benefits to this approach. First, you reduce the amount of space the application requires on the user machine. Second, you reduce the time required to update custom DLLs should the user find an error in it. All you have to replace is one copy of the DLL on the server. There are also two downsides to this approach. First, the user has to have a network connection to make the technique work at all. Second, you could end up increasing network traffic substantially if the DLL is loaded and unloaded frequently.

Absolute Preferences

A shareware or shrink-wrap application author can't assume anything about the application environment — not a source or a destination for the various application files. The machine is total mystery as well. In fact, even the operating system is a

mystery in some cases. You don't have that problem. All you really need to do is worry about the differences between the machines on your network. If you set up all the machines about the same way, you could assume a default destination. Using UNC paths means that you can always assume an absolute source because the server path name won't change from installation to installation. You may even be able to assume a certain amount about the workstation itself, if your network is small and all the machines have similar capabilities. In sum, all this means that you don't have to present the user with as many installation choices and that you can substantially reduce the complexity of the installation program. All it takes is a little preplanning on your part.

No Configuration Choices Needed

Just about every shrink-wrap program offers a choice between three installation configurations: Custom, Typical, and Compact. The Custom choice allows you to choose specific program elements; the Typical configuration is designed for desktop users; and the Compact configuration is intended for people who own laptops. In most cases, you can limit your configuration options to two choices: Laptop and Desktop. In fact, you may want to simply use those terms to keep user confusion to a minimum.

Shareware

Shareware programmers probably have the most challenging job when it comes to packaging their application. Consider a main problem for a shareware programmer: installation size. Unlike the enterprise (corporate) developer who has a high-speed network connection to use, or a shrink-wrap developer who can distribute an application on CD, the shareware developer usually has to make do with a low-speed modem connection to a BBS, online service, or the Internet. Suffice it to say that if your application exceeds 1MB, the number of people downloading it is likely to drop drastically.

The file sizes in this section are meant as guidelines only. The amount of disk space or time that a potential user is willing to invest in your product depends on a good many factors, including perceived value and the current level of exposure you have. For example, one of my favorite shareware graphics packages takes up a whopping 7MB of hard disk space and is well over 1MB in size when compressed. People still download it and set aside the space it requires because this program is well worth the investment. Because it has been around for a while, word-of-mouth tends to give people a reason to download this file as well. What you need to consider is whether most people would be willing to pay the price to use your application—it's something you learn by experience if you're serious about participating in the shareware market.

OK, so you're a little limited on space. How do you get around this problem? The key is in how you market your product. Most of the successful shareware products I've seen use the same key graphics and sounds over and over again.

In other words, instead of coming up with one really fancy graphic for the installation program and a totally different graphic for the application itself, the shareware programmer is content to use the same graphic for both the installation program and the application. In fact, using subtle programming techniques could allow you to get away with using the application's icon in several places.

Can a shareware developer get by without fancy graphics or sounds like the enterprise (corporate) programmer does? Not likely. If you really want someone to pay for the application you've created, you need to add a little polish to it. No one will pay for something that looks drab, even if it does provide much needed functionality. Obviously, trying to weight the space needed by a feature and the amount of pizzazz it provides is difficult.

Another problem that a shareware developer runs into is that of resources. I know of many shareware developers who start out as one- or two-person shops. Many of these shops also deal with consulting jobs and other moneymaking ventures as they wait for a shareware product to, hopefully, take off. Suffice it to say that time isn't on your side, and that it's unlikely that you'll have a professional artist or sound person at your disposal to create the multimedia presentations provided by larger companies who create shrink-wrap applications. Most people are normally satisfied if they see a shareware product that's well designed, space conscious (consumes 5MB or less of hard disk space), and provides at least a modicum of polish.

We haven't yet discussed the biggest problem for shareware developers, and the installation program is the first place you have to deal with it. Enterprise programmers have the most control over their environment by virtue of the fact that they have personal access to every machine that will use the program they create. Enterprise developers can get by with a minimum of machine checks and configuration options. Small to medium-sized business and shrink-wrap developers come next. The small to medium-sized business developer has an intimate relationship with the customer that allow at the to make some assumptions. The shrink-wrap developer can print a set of requirements on the box to ensure that no one uses their application with insufficient hardware. That the user is paying for the application tends to increase his or her awareness of the hardware requirements for installing it. The shareware developer has no such guarantees. A user of your program could have just about any kind of machine ever made — even an old 8088 PC.

What does this lack of control mean? First, that you have to build extra detection routines in your installation program to ensure that minimum hardware requirements are met. Users will rarely read the README file you provide (the one stating the minimum requirements for using the application), and are even less likely to pay attention to them if they do. When the installation fails, you can be sure the user is going to blame you, not his or her own lack of attention span. As a result, you need to build in some type of detection. You also need to build flexibility into your application with regard to configuration.

For example, you may decide to allow the user to configure the application for text mode only and forgo those fancy graphics or sounds on an older technology machine.

The installation program is a good place to sell your shareware product if you have the time and resources to design it properly. Make sure users understand what they'll get in return for buying your product. In other words, because you have a captive audience during the installation process, you may as well use the time to sell your program.

Just like anyone else, the shareware developer does have a few tricks up his or her sleeve to make setting up the installation program easier. Although the following list isn't inclusive of everything you could try, it does provide some ideas of what could work. You need to try a variety of packaging techniques with your application before you will find one that works all of the time.

Granular Packaging

One of the ways to get around the hard disk space and download time problems for a shareware developer is to package your application in several pieces. For example, you could place the main program in one package, the graphics in another, and the sounds in a third package. The result is that users can choose what level of support they're willing to pay for with regard to download time and hard disk space. This concept doesn't come without a price, though. You have to write your application so that it can work without the graphics and sounds (or whatever elements you decide to place in a separate package). Your installation program has to provide similar flexibility. It has to know what do to if a user decides to download one packaging element but not another.

Amplified Help

Creating two sets of help files for your users (one for trial package users and another for registered users) is one way to reduce the trial package size. (Needless to say, this technique also tends to encourage the trial user to register the product.) One of the ways to do this is to make a main and an amplifying help file. The main help file contains explanations for basic commands, whereas the amplifying help file contains user tutorials, macro language descriptions, and detailed command descriptions. Again, the user could decide what level of help to pay for in the form of download time and disk space.

You could also use this feature to your benefit. A shareware package is never guaranteed to sell. Making such a large investment in time without any guarantee of a payback is difficult. Using this approach allows you to make a smaller initial investment until you can see whether a particular shareware product is worth fleshing out.

There are more than a few shareware companies that have used modularized programs to enhance sales. For example, ButtonWare usually provided a simple help file and most of the features for a shareware application when you download it. Buying the application entitled you to download the full-featured program and complete program documentation. Many people are upset by what they call "crippleware" because it doesn't allow them to fully test the application before they buy. Unfortunately, unless a shareware developer provides the user with a good reason to buy their product, most people won't. (You can test the veracity of this statement anytime by looking at the number of shareware companies that die due to a lack of sales even if downloads for their products are brisk.)

Small-to-Medium-sized Business

From a packaging perspective, a small to medium-sized business is a mix between the enterprise and the shareware model. These businesses won't have much in the way of complex equipment and won't normally have a programmer on staff. More likely than not, they won't have a large network either and will instead use workgroup computing to get the job done. In fact, many of these businesses rely on the vertical market applications that a consultant will customize to meet their specific needs. In other words, the application hasn't been written from scratch. Needless to say, this environment both simplifies the packaging requirements (the consultant will probably know every machine on the premises) and makes it harder for the programmer to customize a package at the same time.

Like the enterprise developer, the small to medium-sized business developer can get by with very few graphics. However, because you're developing this application for multiple customers, you need to add a little more pizzazz than an enterprise programmer can get by with. In addition, unlike any other application packager, you might need to develop some unique graphics to go with your package, such as the individual company logos for each company that you deal with. It's important to realize, though, that your audience isn't expecting you to create an application that provides a lot of bells and whistles — they're usually happy if the application works as advertised.

The small to medium-sized business developer also shares one other problem with the shareware developer. The type of target platform will likely vary a great deal because a small to medium-sized business typically can't afford to update their equipment as often as a large company can. So, although you probably won't have to worry about pre-80486 computers, you still have to deal with equipment that's a lot slower and older than most developers have to deal with.

Working in the small-to-medium-sized-business environment has some advantages that aren't found in other markets. For example, you have a closer relationship with all of your clients, which means that you can better understand their needs. This close relationship also means that training is easier. A small-to-medium-sized business developer can create a training program that will likely be more effective and consume less development time than any other developer category.

Obviously, the small-to-medium-sized business development environment is different from the other environments talked about so far. Fortunately, there are a few ways that small-to-medium-sized business developers can make their job easier. Though the following list probably doesn't include everything you've tried, it does provide a few ideas on how to develop packages that reflect your client's needs.

- **Unique Code Included:** There are times when you won't want to give your customer all of the source code for a custom application. In fact, there might be times when it would be illegal to do so because you might need to include licensed source code. However, in most cases, you'll want to provide the user with a package that contains the unique source code for his or her custom implementation of a generic application. Make sure that you provide this code in a separate directory so that it won't get loaded on the client machine.
- **Personalized Support Dialogs:** Unlike the enterprise programmer, it's likely that you'll service the needs of several clients. What this means is that you need to provide better support information in your application package. Make sure you include support information as part of the installation program, README file, all help files, and within the application itself.
- **Customized License Agreement:** Enterprise programmers don't need to provide a license agreement for their software because it's written for a specific company and the licensing agreement (when there is one) is defined within the contract. A shrink-wrap or shareware developer will normally include a generic licensing agreement that applies to everyone equally. Only the small to medium-sized business developer is faced with a situation where the licensing agreement for a generic application could differ from company to company. Make sure you include this licensing as part of the package in both the installation program and within the application's About dialog box.

Shrink-Wrap

I'm not going to pretend to tell a company like Microsoft how to market or package their product — the effects of their sales force are already legendary, and I doubt that I can do much to help them out. One of the things that usually sets a shrink-wrap product apart from shareware is the size of the company producing the application. Larger software companies usually concentrate on one or more products, and they have a large professional group of people to help put the packaging together. However, there are a few things that the average programmer can learn by looking at these shrink-wrap packages.

During the past few months, I decided to start taking notes whenever I installed a shrink-wrap or shareware product. After installing about 30 products, I decided to take a look back over my notes. The results were surprising in some cases. For example, the previous section mentions using the installation program as a means to sell your product if you're a shareware vendor because you have a captive audience and fewer sales resources at your disposal. It may surprise you to learn that shrink-wrap software commonly uses the installation program for this very purpose, but in a different way than a shareware vendor would use it. The user has already purchased the product being installed, but how about add-

on products? Shrink-wrap vendors commonly use the installation program to sell some add-on product that relates to the application the user is installing (one example is an extended dictionary for a word processor).

Another thing I noticed is that shrink-wrap software commonly tells the user what's new about the program during the installation process. It doesn't surprise me much because the shrink-wrap companies must have figured out long ago that most users don't read the README file and the installation program is usually the last part of the product to get finished. Again, the shrink-wrap vendor has a captive audience, so they use the installation program to give the user an overview of what they would learn if they'd actually read the README file.

It should also come as no surprise that the shrink-wrap installation programs are usually packed with all kinds of multimedia presentation materials. After all, the shrink-wrap vendor has the resources required to produce such a display. Even if you don't have the resources to duplicate the presentation provided by a shrink-wrap vendor, you can make notes, as I have, to figure out what works and what doesn't. Providing a smaller version of the same type of presentation in your installation program is a sure way to make it look more polished. Remember that you want to give users the best possible impression of your product from the outset to ensure they enjoy working with it. Creating a nice-looking installation program won't do a single thing to make your application work better, but it'll affect the user's perception of your application, which is a very important part of getting the user up and running with a minimum of support.

There are some negative lessons to learn from shrink-wrap vendors as well. One of the best things I learned was that you can make your installation program too complex to test thoroughly. I recently tried installing one product and found that the help screens for the program were for the previous version. The vendor had forgotten to update the screens as needed for the new version of their product. The result? Because the product didn't come with any printed documentation, I didn't have a clue as to whether I should install certain product features. If the vendor had taken a few minutes to fully test the installation program, I would have had all the information needed to make an informed decision about which product features to install.

Another very strange problem occurred when the product failed to install at all. It seems that the installation program used special graphics that relied on a product feature that was incompatible with the machine I was using. Moving the product to another machine where the installation program worked just fine showed that the product would work, even though the installation program wouldn't. Again, the vendor decided to add some really neat special effects to the detriment of the program as a whole.

As you can see from this section, looking at what other people are doing in the way of packaging a product can save you time and effort when it comes to packaging your own product. I find that taking notes and then reading them right before I write my installation program keeps me from making as many mistakes. (Very few people get an installation program right the first time.) Obviously, you

want to spend a lot of time testing the installation program on a wide variety of machines to ensure that it works the way that you envisioned it would. Remember that the installation program is the very first thing that the user will see, so testing your installation program is at least as important as testing the application as a whole.

Determining Which Packaging Model to Use

The previous section outlined four different packaging schemes based on common software usage environments. Unfortunately, though these four scenarios do cover a relatively large percentage of the development efforts out there, there may be times when the application you've developed will be used in more than one of these environments.

Here's an example. Suppose you write an accounting application that has vertical market add-ons. Many users will simply buy the shrink-wrapped version of the standard package because they don't require any additional functionality. These people would fit the shrink-wrap model. In other cases, your customer may be a developer who integrates the general package, one or more of the vertical market add-ons, and then adds some extra value in the form of custom modules. This person could fit into either the enterprise or the small to medium-sized business model. As you can see, there are times when the application you develop might not fit in one packaging model, so it's important to provide some flexibility in the packaging system you design.

There are other considerations when it comes to choosing a packaging model. Not every situation requires a large multistep installation program and associated help files. The following list provides some ideas on what factors to consider when creating your application package.

- **Application Size:** The size of your application determines just how much you need to add in the way of additional support. For example, if you're creating a package for a Notepad-style editor, you don't need an extensive help system as you do with a Word-style editor.
- **Application Complexity:** Don't confuse application complexity with application size. You can create a small application, like a network-monitoring utility, which performs a lot of complex tasks but isn't very large. Complexity is a measure of how much knowledge a user requires to use the application, how many features the application provides, the depth of the menuing system, and the overall number of windows the user will need to work with. The more complex the application, the larger the help file needed and the more options you need to add to the installation program.
- **Application Modularity**: Utility programs are normally the least modular application you can create. In most cases, they contain two modules: the application and an associated help file. On the other hand, major applications can contain literally hundreds of modules. The application package you create

should reflect the modularity of the application. More modules mean more options in the installation program. Make sure you include options to tailor the help system to reflect the number of modules the user chooses to install. If you choose to implement application advertising (discussed later in this chapter), make sure you also advertise the modularity of your application so that the user understands that they don't need to install everything to get the level of required functionality.

- **Audience:** A novice user is going to require more handholding than will an experienced user. Extensive bullet help, higher levels of modularity, reduced complexity, and the use of wizards all help the novice user. You may even want to include a "simple" or "starter" mode that reduces your application's feature set to those that the novice can handle. On the other hand, some (if not most) experienced users will feel that you're talking down to them if you overwhelm them with help. This particular concern is so important that some applications actually take user level into account (as indicated by the user) when installing and configuring the application. A novice user might only gain access to a few very important features, whereas an experienced user will have access to everything the application has to offer.
- **Target Platform:** The kind of hardware and operating system that your application will run on often determines the number of special effects that you add to the application package. Obviously, you don't want to provide multimedia presentations on an old 80486 machine. Likewise, you don't want to make your application package appear outdated by using text-only displays on a modern machine with the latest operating system installed.
- **Network Functionality and Internet Capability:** Networks and the Internet present special challenges when putting the user an application package together. You might need to include special aids to help get over potential network problems. In addition, your installation program may need to check for the presence of Internet support prior to installing specific Internet features.
- **Special Hardware or Hardware Specific:** A lot of applications rely on special hardware. For example, many graphics applications need a scanner attached to the target machine before they can grab images from print. Drivers are always a concern when it comes to special hardware. There have been more than a few situations where the application program I've installed also installed the latest version of the vendor's drivers for me to ensure that the application will work as intended.

What to Include in the Package

Never get the idea that you can just slap an installation program together in a few seconds without doing any research. Any program that runs in the Windows environment is a lot more complicated than even the programmer realizes in most

cases. For example, all of the examples in this book have relied on the C runtime files. Yet, I've never mentioned adding these files to your application code, nor have you seen them referenced in any of the source code. That's because these files are automatically added to your SYSTEM folder when you install Visual C++ in the first place. You don't need to know about them while you design your application because their inclusion is automatic. The same can be said of the MFC library files and those required for database applications. In most cases, Visual C++ takes care of these file additions for you without much, if any, thought on your part.

However, now you're trying to put everything needed to run your application in a package for the user. At this point, in the product development cycle, you no longer have the luxury of ignoring those other files — they're part of your application and must be included with the package sent to the user. The problem is figuring out which files to include with your program.

Figuring out which DLLs you need to ship with your application is a time-consuming task, especially if you resort to trial and error to figure things out. However, there are three methods you can use to take some of the trial and error out of the process:

1. You could use the Windows QuickView utility to view the Import Table entries for your application. The Import Table list tells you which DLLs your application relies on. Figure 22-1 is an example of a QuickView list for the SDI example that we looked at in Chapter 3. Notice that it not only shows which files your application uses, but the function within those files as well.

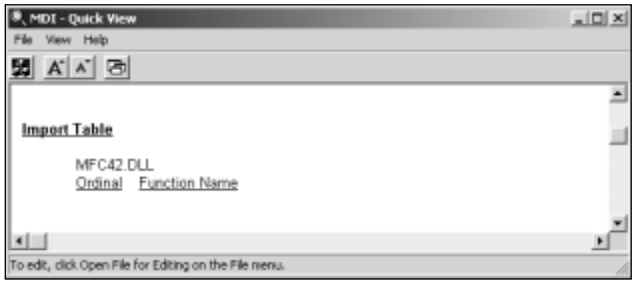
MDI - Quick View
File View Help
Import Table
MFC42.DLL
Ordinal Function Name
To edit, click Open File for Editing on the File menu.

Figure 22-1: The Windows QuickView utility provides one way to look for the files your application depends on.

2. The DumpBin utility can give you a DOS-command-line view of your application's requirements. I've found that DumpBin is a little more trouble to use, but it often finds information that's hidden from the QuickView utility. You'll also find that it offers more options when it comes to the kinds of information to find. All you need to type at the command line to use this program is DUMPBIN /IMPORTS SDI.EXE. Figure 22-2 shows what the output of this program looks like (it's a command-line utility).

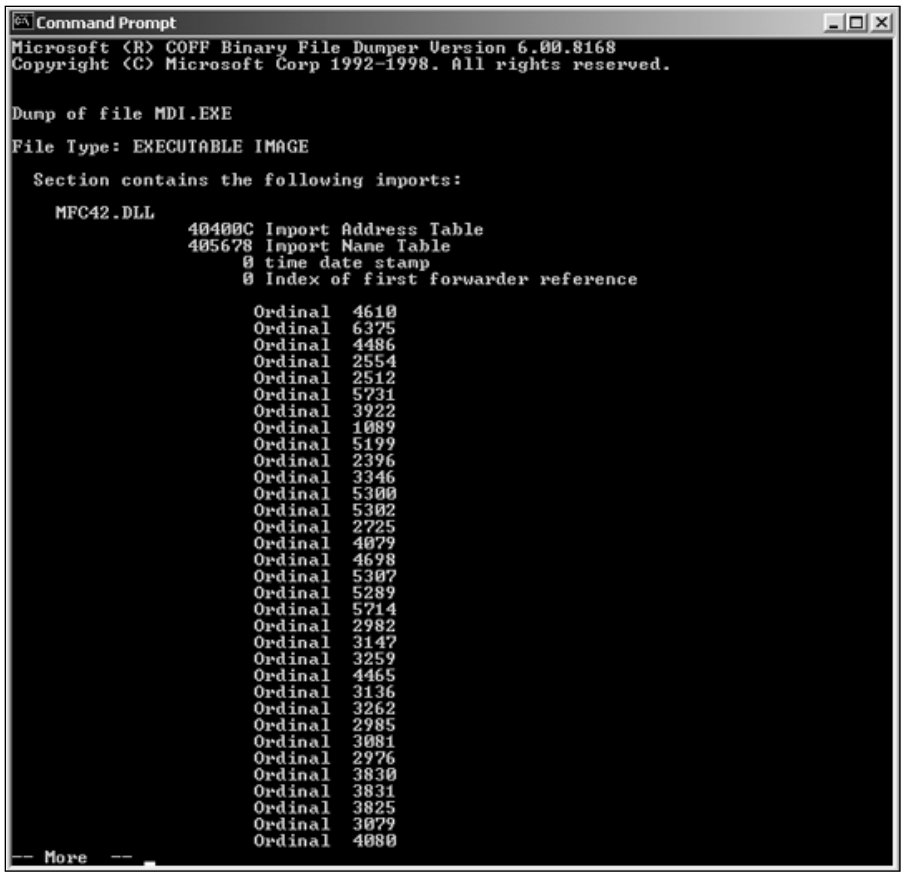

Figure 22-2: There are times when DumpBin will show you information that's hidden from QuickView.

You may get a missing-file message when using the DumpBin utility. Both the MSDIS100.DLL and MSPDB50.DLL files are located in the Program Files/DevStudio /SharedIDE/bin directory on your machine (assuming you've used the default settings during installation). You need both of these files along with the LINK.EXE found in the VC/bin directory to make the DumpBin program work.

3. If you have the latest version of the Windows SDK, you'll find the Depends utility, which you can use to get the DLL requirements for your program. Figure 22-3 shows the Depends utility at work. Notice that the Depends utility provides a lot of information not provided by the other programs. For example, you can see the actual hierarchical relationship between the dependent file. The Depends utility also provides essential version information. In sum, though the Depends utility display looks a bit more complex, you'll also find that it's more informative.

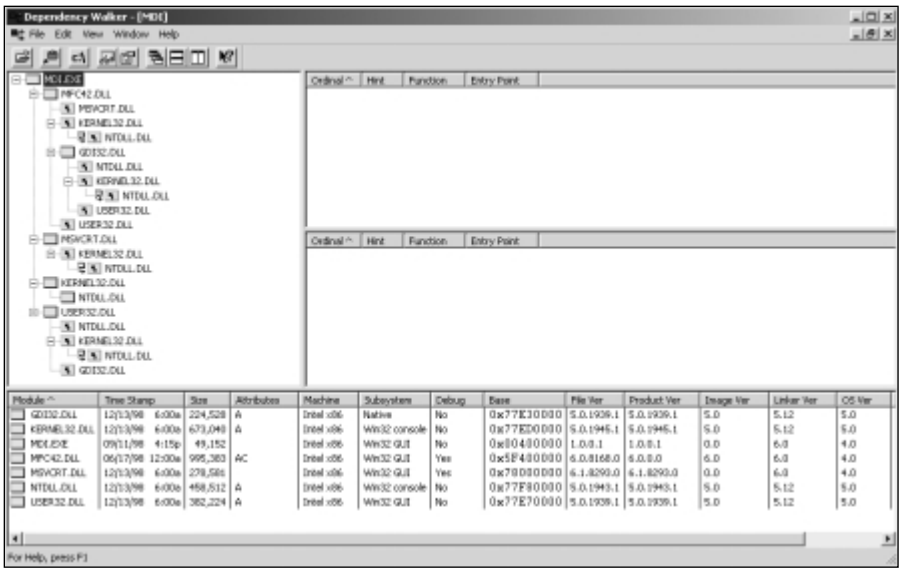

Figure 22-3: Of all the ways to display a list of files that your application requires, the Depends utility is the most comprehensive.

Getting a list of files that your executable uses directly isn't the end of the process. You'll also need to check the DLLs it uses for any dependencies they may have as well. Obviously, you can ignore files such as USER.EXE because the user will have them installed as part of Windows itself, but you need to watch for files that may not have been installed. In short, manually check each file for it's dependencies, and then include those files with your application.

Check a clean machine for a list of files that a specific version of Windows provides by default. You may want to maintain a list of these files for later reference if you plan on creating more than one installation program. In addition, be sure to keep separate lists of the files that each version of Windows that you want to support contains. For example, the files in the OSR2 version of Windows 95 differ from those found in the OSR2.5 version, both of which differ from those in Windows 2000.

Once you have a complete list of the files you need for your application, gather them all in one place. Copy this set of files to a newly installed version of Windows (whichever version your application is targeting). Test your application to make sure it works. You'll get messages if you're either missing a file or it needs to go into the SYSTEM folder. Make sure you keep track of the files you place in the SYSTEM folder because you'll have to place them there on the user's machine as well.

Always try to keep all of the DLLs for your application in the application folder instead of automatically placing them in the SYSTEM folder. Keeping everything in one place will help you uninstall the program later should the user want to remove it from his or her machine. You'll also find that you have fewer problems with corruption from other programs that the user installs—for example, when an older version of the same DLL used by another application overwrites a DLL you need to run your application.

Using InstallShield

At this point, you should have figured out how you want to package your application, which includes deciding on which packaging model—small-to-medium-sized business, enterprise (corporate), shareware, or shrink-wrap—to use. Make sure this packaging strategy considers application particulars such as size, modularity, and complexity. In addition, your application packaging strategy should include the type of user and the computing platform.

CD-ROM Location: Chapt22\My Installations\Single Document Interface Application Example Program Name: Setup.EXE Source Code Modules in Text: Setup.RUL

You should also have a complete list of the files needed to run your application. Don't forget hidden files such as the MFC runtime when needed for your application. Make sure your file list includes specific installation areas for each file. For example, note which files have to install in the SYSTEM folder. File version numbers are important as well, especially when it comes to Visual C++ or application-specific DLLs that may already appear on the user's machine, but in an older format. Finally, you should have tested your application and associated files on a clean machine—one that has had only Windows and any appropriate service packs installed. It doesn't pay to start writing an installation program until you're certain you've included all of the files it requires and that the setup works as anticipated on a clean machine.

This section relies on the version of InstallShield provided with Visual C++ 6.0. Earlier versions of Visual C++ may not have this version of InstallShield, which means you'll probably need to modify the procedures in this section to meet your needs. Even if you're using the same version of InstallShield outlined here, you may see slightly different screens or have a few different options depending on the application you're trying to package. In short, it's not unusual to see a few variations from what's presented here when using InstallShield. You need to install the InstallShield product—it's located in the IShield directory on your Visual C++ 6.0 CD. The procedures in this section assume that you've already installed the InstallShield product on your hard drive.

It's time to create a sample install program using a typical end-user application as a starting point. So far, we've added some resources (Chapter 4) to the SDI application we created in Chapter 3. Now it's time to package this application to send to someone else—well, at least for example purposes.

The SDI example isn't really complete, but it's complete enough to use for demonstration purposes. To create a complete package you need to develop a help file, a README file, a tutorial, and perform extensive testing. In fact, these are just the minimum supplements that you'd normally include. There's a wealth of other inclusions you might want to add, such as sample files, additional utility programs, a network administration program, and the like. Make sure you have the complete package put together before you create the installation program.

The following procedure shows how to put a typical installation program together. We won't do anything really fancy; the whole idea is to get the application ready to send to someone else. If you were to pick a specific packaging model for this example, it's very close to the enterprise (corporate) packaging technique that we talked about earlier.

We discuss a generic installation program in this section; you'll need to modify it to meet the needs of your particular application and installation environment.

Try This

1. Start the InstallShield 6 Visual C++ 6 Edition program. You'll see a window similar to the one shown in Figure 22-4. No project is defined yet, but that there's an Installation Wizard entry in the Projects window. This window doesn't look precisely like the one in Visual C++, but as the procedure goes on you'll notice certain similarities that tend to ease the installation program creation process. Notice also the InstallShield link in the lower-right corner. Clicking it will open your browser and take you to the InstallShield online help site on the Internet.

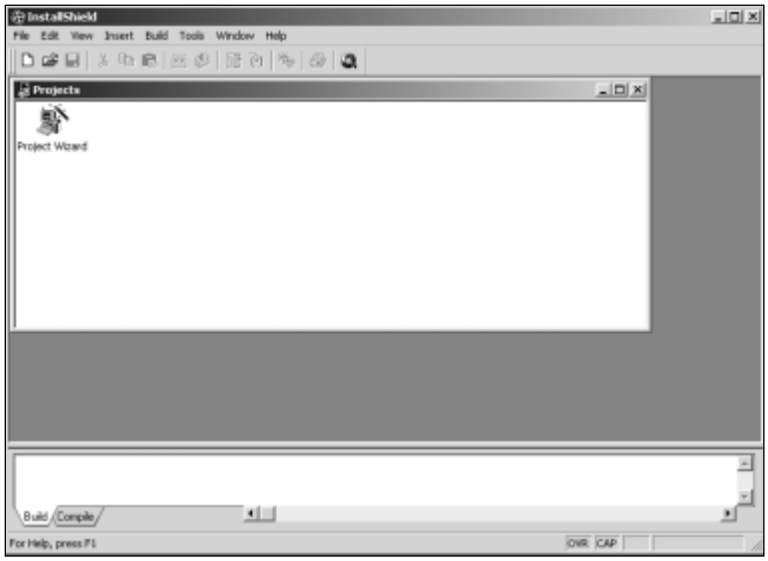

Figure 22-4: InstallShield allows you to create an installation program using techniques similar to those you've already used in Visual C++ to create your application.

2. Double-click on the Project Wizard icon in the Projects window. You'll see a Project Wizard – Welcome dialog box similar to the one shown in Figure 22-5.

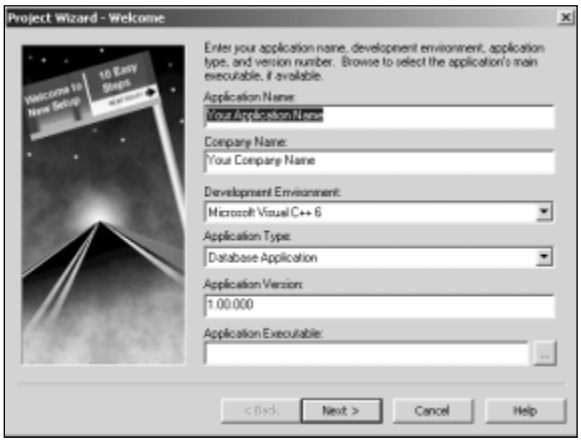

Figure 22-5: This is the first dialog box you see when creating a new project.

The Professional Edition of InstallShield offers advanced features on the Welcome (and other) page—make sure you read about the enhancements in the help file.

3. Type the name of your application, such as **Single Document Interface Application Example**, which is shown in the example.

4. Type your company name. The example program uses **A Sample Company**.

The Visual C++ 6 Edition of InstallShield provides only one Development Environment field entry, Microsoft Visual C++ 6. The professional version supports a much broader variety of development environments, making it much more flexible. The development environment support determines what languages you can use with InstallShield.

5. Choose one of the entries in the Application Type field. The example program uses Software Development Application—there isn't a standard utility program type, and the Software Development Application type does allow for utility programs.

6. Type the application version number in Application Version field. Our example program uses **1.0**, but you can use any numbering scheme consistent with your company policies.

7. Click the . . . button next to the Application Executable field. You'll see a standard Open dialog box that you can use to find the application you want to package on your hard drive. The sample application uses SDI.EXE (the version in the \Win_NT\Chapt04\SDI\Release\ folder on the CD that we created in Chapter 3 and modified in Chapter 4).

8. Click Next. You'll see a Project Wizard – Choose Dialogs dialog box like the one shown in Figure 22-6. This is where you choose the series of dialog boxes that the user will see when installing your application. Because we're using the enterprise model for our installation program, I unchecked the Software License Agreement, Setup Type, and Custom Setup dialogs. You'd choose some or all of these dialog boxes for the other packaging models we discussed in the "An Overview of the Packaging Models" section of the chapter. There are a few things that you should notice about this particular dialog box. First, look in the lower-left corner as you move from one dialog box entry to the next. The Project Wizard gives you a thumbnail view of the dialog box in question to make it easier to determine whether you actually want it. Highlighting a dialog box and then clicking the Preview button will display a dialog box that looks exactly like the one the user will see. Obviously, you'll be able to modify this dialog box later, but it helps to know what you're starting with.

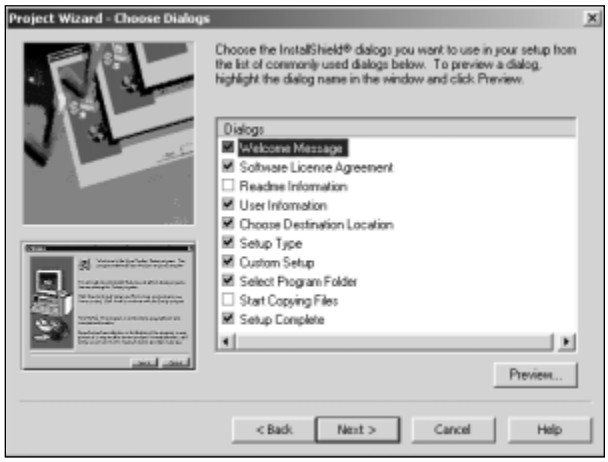
Project Wizard - Choose Dialogs
Choose the InstallShield® dialogs you want to use in your setup from the list of commonly used dialogs below. To preview a dialog, highlight the dialog name in the window and click Preview.
Dialogs
Welcome Message
Software License Agreement
Readme Information
User Information
Choose Destination Location
Setup Type
Custom Setup
Select Program Folder
Start Copying Files
Setup Complete
Preview...
< Back
Next >
Cancel
Help

Figure 22-6: This dialog box is one place where your choice in platform comes in play—it allows you to choose the installation dialog boxes the user will see.

Clicking one of the entries in the Dialogs list will display a thumbnail display of that dialog box in the lower-left corner of the Project Wizard – Choose Dialogs dialog box. Unfortunately, some of these dialog boxes are still very difficult to see. Obviously, this makes it hard to know if you actually want to use that dialog box in your installation program. If you want to see a full-sized presentation of the dialog box in question, click Preview. Using this button will allow you to see what the dialog box will look like to the user, sans any personal or company information that you'll eventually add to it.

9. Choose which dialog boxes you want to add to the installation program, and then click Next. You'll see a Project Wizard – Choose Target Platforms dialog box like the one shown in Figure 22-7. Because we want to be able to install our program on any of the supported programs, we won't have to make any

changes — just make sure all of the platforms are highlighted. (Figure 22-7 doesn't show any of the platforms highlighted to make the text easier to read; the dialog normally starts out with all of the available platforms highlighted.) However, there may be situations when you want to be a little more selective. Reducing the number of supported platforms accomplishes two things. First, it reduces the size of the installation program you create, which is a real plus for shareware developers who are short on space to begin with. Second, it reduces the chance that someone will try to use your program under the wrong version of Windows, which is also a real plus for any developer.

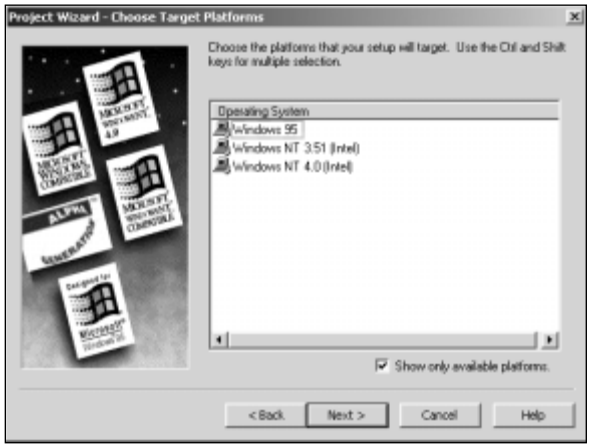
Project Wizard - Choose Target Platforms
Choose the platforms that your setup will target. Use the Ctrl and Shift keys for multiple selection.
Operating System
Windows 95
Windows NT 3.51 (Intel)
Windows NT 4.0 (Intel)
Show only available platforms
< Back
Next >
Cancel
Help

Figure 22-7: Make sure you add all of the platforms you want to support because adding them later can be difficult.

It's educational to uncheck the Show Only Available platforms checkbox to see all of the choices offered by the Professional Edition of InstallShield. As of this writing, there aren't any Windows 2000-specific options for the Visual C++ 6 Edition of the InstallShield program. The Windows NT 4.0 setting worked just fine for the sample application, although your experience with this setting may vary. The best choice is to either upgrade your copy of the product when a new version of InstallShield appears on the market or wait until Microsoft updates Visual C++ 6 with a new service pack.

10. Select one or more platforms as needed (but at least one), and then click Next. You'll see a Project Wizard – Specify Languages dialog box like the one shown in Figure 22-8. Because the Visual C++ 6 Edition supports only one language, we don't have to make any changes here (make sure English is highlighted, even though Figure 22-8 doesn't show it that way). As with other selections that you've had to make so far, more isn't necessarily better. Adding languages increases the size of the installation program and could add user confusion as well; choose only the languages you need.

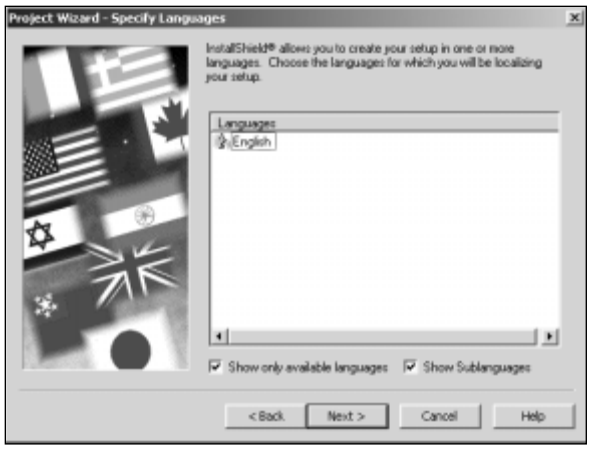
Project Wizard - Specify Languages
InstallShield® allows you to create your setup in one or more languages. Choose the languages for which you will be localizing your setup.
Languages
English
Show only available languages
Show Sublanguages
< Back
Next >
Cancel
Help

Figure 22-8: InstallShield can provide support for creating installation programs in more than one language.

11. Highlight one or more languages (but at least one), and then click Next. You'll see a Project Wizard – Specify Setup Types dialog box like the one shown in Figure 22-9. Because we're using the enterprise packaging model, I've already chosen the Network setup type. Notice that there's a standard network type and two special network types; the first is optimized for space, whereas the other is optimized for performance. Any of these choices would work well in the enterprise environment. Other packaging models require that you make other selections, including the most common of all: Custom, Typical, and Compact. It's interesting to note that InstallShield does offer a wealth of other choices, though.

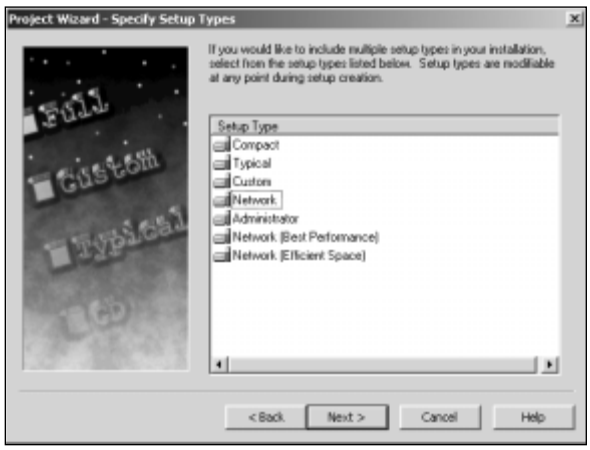
Project Wizard - Specify Setup Types
If you would like to include multiple setup types in your installation, select from the setup types listed below. Setup types are modifiable at any point during setup creation.
Setup Type
Compact
Typical
Custom
Network
Administrator
Network (Best Performance)
Network (Efficient Space)
< Back
Next >
Cancel
Help

Figure 22-9: Make sure you choose at least the three common installation types for complex applications.

12. Choose one or more setup types, and then click Next. You'll see Project Wizard – Specify Components dialog box like the one shown in Figure 22-10. The Project Wizard doesn't assume anything about the components that you want to install. The example program uses Program Files and Shared DLLs. You need to choose each of these options one at a time to define the files that go under each component type. A component type is a definition of a major program area. For example, the user may want to include all of the sample files, but none of the help files. A component type doesn't necessarily define which files you need to perform the task, simply that the task is a separate entity that the user could either choose to install, or perhaps that needs to be installed in a different directory from the rest of the files.

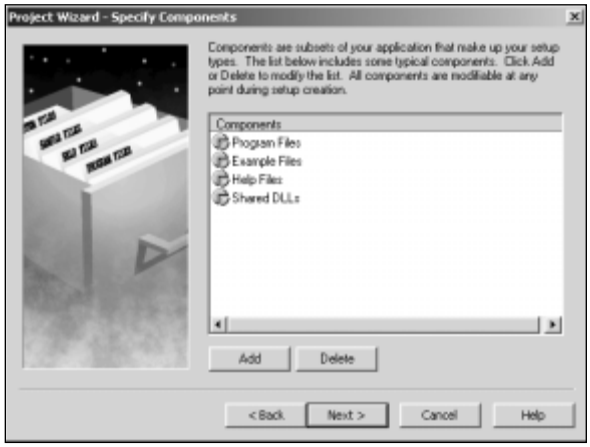
Project Wizard - Specify Components
Components are subsets of your application that make up your setup types. The list below includes some typical components. Click Add or Delete to modify the list. All components are modifiable at any point during setup creation.
Components
Program Files
Example Files
Help Files
Shared DLLs
Add
Delete
< Back
Next >
Cancel
Help

Figure 22-10: The modularity of your application determines the number and type of components for which you need to create installation options.

You can add new component types as needed to your installation program. For example, you might need to add a Database File component if you've written a database application. All you have to do is click Add and the Project Wizard will add the new entry for you. Type the name of the component and press Enter to complete the addition process.

13. Highlight Help Files, and then click Delete. Highlight Example Files, and then click Delete. Project Wizard will remove these component types from your installation program.

14. Add or delete component types as needed and then click Next. You'll see a Project Wizard – Specify File Groups dialog box like the one shown in Figure 22-11. Our example program uses Program Executable Files and Shared DLLs file groups. Normally, you'll create file groups so that all of the files needed

for a specific task are copied at one time. File groups can cross component type boundaries. For example, say the spell checker and the grammar checker both rely on the same set of DLLs, but require different rule files. You could define a common file group for the DLLs and two other file groups for the rule files. If the user selected the spell checker but not the grammar checker, InstallShield would copy the spell checker file group and the common DLL file group, but not the grammar checker file group.

Project Wizard - Specify File Groups
File groups are logical sets of the files that make up your application. During setup creation, file groups are associated with components.
The list below includes some typical file groups. All file groups are modifiable at any point during setup creation.
File Groups
Program Executable Files
Program DLLs
Example Files
Help Files
Shared DLLs
Add
Delete
< Back
Next >
Cancel
Help

Figure 22-11: Groups allow you to create general file classifications that make it easier to determine if you've included everything for a specific kind of install later.

15. Highlight Program DLLs and then click Delete. Highlight Example Files and then click Delete. Highlight Help Files and then click Delete. We've just eliminated the three file groups that we won't need for the example program. Obviously, the selection of file groups depends on how your application is put together and where you need to copy the files. Remember that all the files in one file group will go to the same directory on the hard drive.

16. Add or delete file groups as needed and then click Next. You'll see a Project Wizard – Summary dialog box like the one shown in Figure 22-12. At this point, you should check the list of options to make sure everything is correct before asking the Project Wizard to build the installation program for you.

17. Click Finish. InstallShield will create the installation program using the parameters you've just specified. At this point, the InstallShield interface will change as shown in Figure 22-13. Notice that you can now see the C++ code required to create the installation program, and that you can modify it just like any other project.

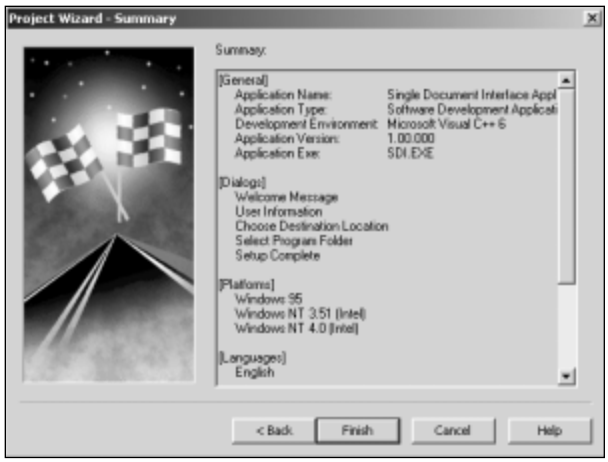

Figure 22-12: The Summary dialog box is the best place to check to make sure that you've included everything needed for the installation program.

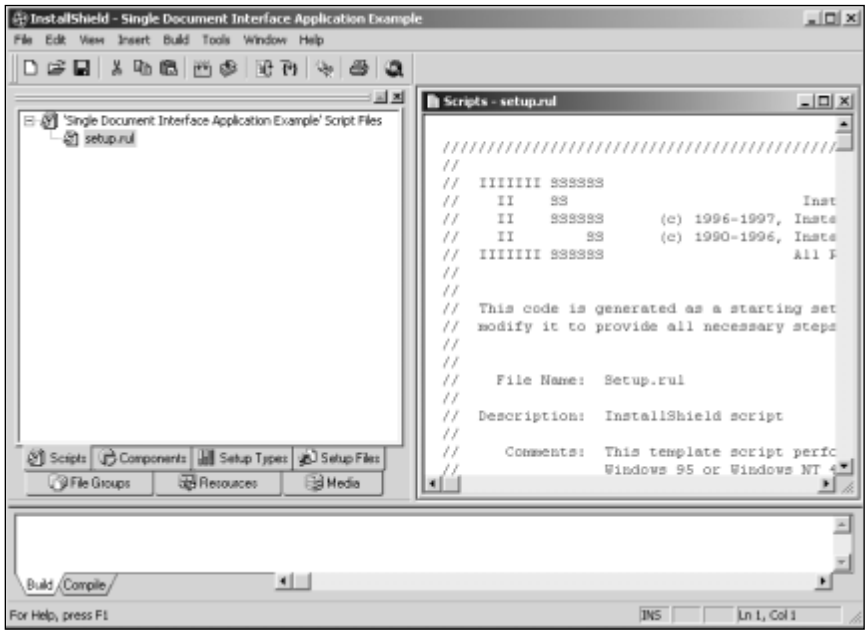

Figure 22-13: Once you've created a project, InstallShield will generate it for you, and then display the resulting source code.

Don't get the idea that InstallShield generates source files that are precisely like the ones used in a general project. There are some ancillary files that you haven't seen in the past such as the setup rule (SETUP.RUL) file shown in Figure 22-13.

At this point, you should have a barely usable installation program shell. There's still quite a bit of configuration to do, though, and we'll look at all of it in the sections that follow. The important thing to realize is that you'll write very little code during this process — InstallShield does most of the work for you as you define the various program elements.

If you've still got questions about how to use InstallShield after reading this section, there are a couple of ways to get more information. This is one of those times when it really pays to look in the product README file, because it contains links to a lot of places you need to know about on the Internet. For example, you'll find a link to the newsgroups site at `http://support.installshield.com/newsgroups/default.asp`. This site contains links to several InstallShield-specific newsgroups. Just clicking one of the links will create a new folder in your newsreader containing the newsgroup you wanted to look at. There's even a special newsgroup for the Visual C++ 6 Edition at `installshield.is5.free-edition`. You'll also find links for newsgroups covering things such as the IDE, scripting, and the use of multimedia. In short, these newsgroups provide you with the contacts you need with other developers interested in finding solutions to packaging their applications successfully.

Setting Up the Components

The first task to accomplish is setting up the various components. Remember that a component is a selection that appears during the installation. If you've ever used the custom install feature of an installation program, then you know what a component is. It's the series of checkboxes that allow you to choose whether to install a particular program feature such as the help files.

Configuring the components for your installation program is fairly easy. The following procedure will get you started.

Try This

1. Click the Components tab and you'll see a Components – Program Files dialog box like the one shown Figure 22-14. The components dialog box contains a complete list of all the values for the selected component. Remember that we have two components in this case: Program Files and Shared DLLs. It's important to set up both components before proceeding to the next section.

Figure 22-14: This dialog box provides an overview of the various component options.

2. Double-click the Description property and you'll see a Properties dialog box like the one shown in Figure 22-15. All of the other properties will show similar dialog boxes. Each Properties dialog box describes what you're supposed to do with this particular property and will also allow you to type in a value for the property.

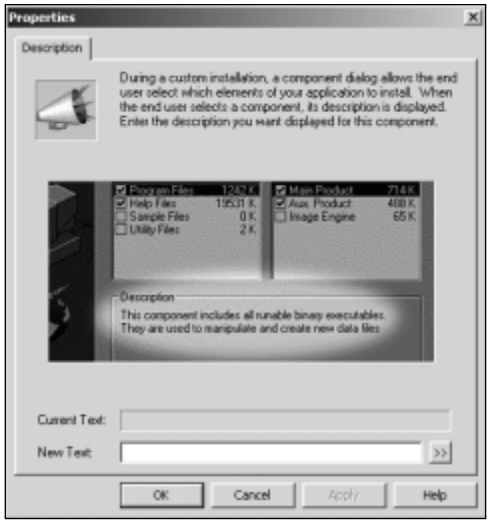
Properties
Description
During a custom installation, a component dialog allows the end user select which elements of your application to install. When the end user selects a component, its description is displayed. Enter the description you want displayed for this component.
Program Files 1425 K
Help Files 1953 K
Sample Files 0 K
Utility Files 2 K
Main Product 714 K
Aux. Product 488 K
Image Engine 65 K
Description
This component includes all runable binary executables. They are used to manipulate and create new data files.
Current Text:
New Text:
OK
Cancel
Apply
Help

Figure 22-15: Use this dialog box to add a description for the component.

3. Type **All of the files required to run the application.** and then click OK. The Description property will now contain the text you just typed. There's one last item that you absolutely must take care of to configure the components. Look at the Included File Groups property. Right now we haven't assigned any to the Program Files (or any other) component.

4. Double-click the Included File Groups property and you'll see a Properties dialog box like the one shown in Figure 22-16. This Properties dialog box contains a complete list of the file groups assigned to a particular component. Remember the example about making the spell checker and grammar checker optional components of your installation. They both use the same DLLs but require different rule files. I mentioned setting up three file groups: one with the spelling rule file, another with the grammar rule file, and a third with the common DLLs needed by both. This is the property where you set up that relationship.

Figure 22-16: The last required configuration option for components is the Included File Group.

5. Click Add and you'll see an Add File Group dialog box like the one shown in Figure 22-17. As you can see, the Add File Group dialog box contains the two file groups we defined using the Project Wizard.

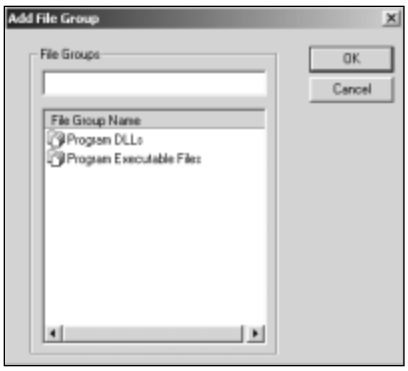

Figure 22-17: The Add File Group dialog box allows you to add file groups to a particular component.

6. Highlight the Program Executable Files entry, and then click OK. InstallShield will add the entry to the Included File Groups tab.
7. Click OK. You'll see the entry added to the Components – Program Files dialog box.

Modifying any other entries follows the same pattern that we followed in this section so far. You'll need to make the following assignments between components and file groups before going on in the chapter. Table 22-1 shows the settings I used for the example program.

Table 22-1
Component Settings for the SDI Installation Program

Component	*Property*	*Value*
Program Files	Description	All the files required to run the application
	Status Text	Copying Program Files. . .
	Overwrite	SAMEORNEWERVERSION/SAMEORNEWERDATE
	Required Components	Shared DLLs
	Included File Groups	Program Executable Files
Shared DLLs	Description	Common files used by the program
	Status Text	Copying Shared DLLs. . .
	Overwrite	SAMEORNEWERVERSION/SAMEORNEWERDATE
	Destination	<WINSYSDIR>
	Included File Groups	Program DLLs

You don't need to change all of the component properties — what you do need to change depends on the kind of application package you're creating. For example, in our enterprise (corporate) model installation, we don't ever allow the user to choose the components that get installed, so providing a component description is pointless. The user won't ever see it anyway. There are some properties that you should consider changing regardless of the packaging model you're using, and the following list looks at each.

- **Status Text:** This is the text that the user will see while the installation program copies the files from the source to the destination. The Progress dialog will say something like "Copying program files. . . ," which works fine for default components. You may want to define something special for custom components.

- **Installation:** If you've ever been annoyed by a program coming along and copying over the new DLLs you just installed with really old ones that don't work, then you'll know why this property is important. It tells InstallShield whether you want to look at the time and date stamp on the component files first before you overwrite them. I always choose NEWERVERSION/NEWERDATE or SAMEORNEWERVERSION/SAMEORNEWERDATE instead of the default ALWAYSOVERWRITE. In essence, these options tell InstallShield to only overwrite a file if the source has both a newer (or same) date and a newer (or same) version than the one on the user's hard drive.
- **Destination:** The standard destination for all of your application files is the target directory, the one the user chose for the program. However, there are some circumstances when using the target directory won't work like it should or could waste space on the user's machine. For example, most Visual C++ applications require the C run-time files and the MFC files. What if every application you installed added these files to a program directory instead of a centralized place? It wouldn't take very long to fill your hard drive with useless files. In most cases, you'll probably want to copy your Shared DLLs component files to the Windows SYSTEM folder. All you need to do is change this property. (When you open the Property dialog box, you'll see a map of the various locations that you can use for copying the file; just choose the one you want.)
- **Required Components:** This is where you'll set up dependencies between components as shown in Figure 22-18. What I'm telling InstallShield, in this case, is that if the user installs the Program Files component, the user must install the Shared DLLs component as well (which only makes sense because without the Shared DLLs the program won't run).

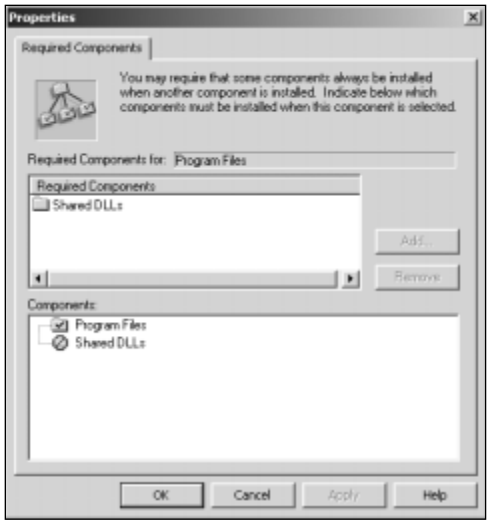

Figure 22-18: Set up component dependencies as part of the process of defining your installation program.

Pay special attention to the FTP Location and HTTP Location properties if you plan to install your application from an Internet or intranet site. These properties allow a user to start the installation from a Web site link, and then copy the needed files from your Web server. You'd normally choose the FTP Location property if the files were under the control of the FTP server.

Setting Up File Groups

Creating a set of components doesn't help the installation program very much. You've told the user what components can be copied to the hard drive, but you still haven't told the installation program what those files are. This is the step where you do that. We'll define a set of files for each of the file groups that we defined previously. The following procedure will get you started.

Try This

1. Click the File Groups tab. What you'll see is a list of the file groups and the File Groups dialog box shown in Figure 22-19. Choosing which file group to configure is easy. Just click on its entry in the File Groups window on the left side of the display.

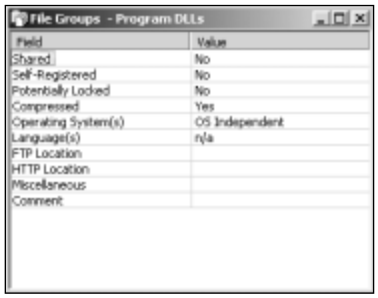

Figure 22-19: The File Groups – Help Files dialog box allows you to configure each of the file groups that we defined previously.

2. Click the plus sign (+) next to the Program DLLs folder in the File Groups window. Click the Links entry under the Program DLLs folder and you'll see the File Groups – Program DLLs\Links dialog shown in Figure 22-20. This is where you'll define a list of one or more files to include for this file group.
3. Right-click the File Groups – Program DLLs\Links dialog and choose Insert Files from the context menu. You'll see a standard File Open type dialog box.
4. Highlight the MFC42.DLL and MSVCRT.DLL files found in the Windows System folder.

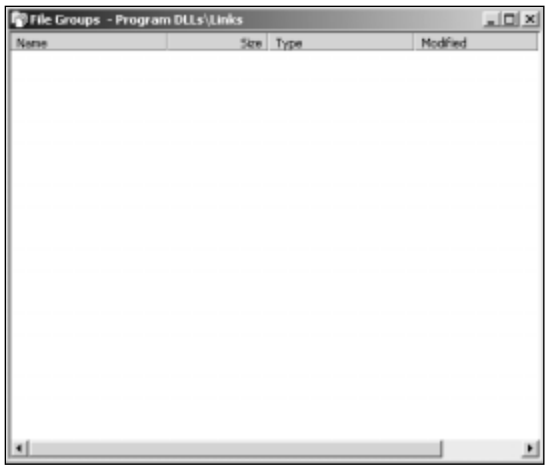

Figure 22-20: The File Groups – Help Files\Links dialog box allows you to configure the file groups.

5. Click OK to add these file to the Program DLLs file group.
6. Perform Steps 2 through 5 to add the SDI.EXE file to the Program Executable Files file group.

Defining Resources

By this time, you should have the components and the file groups set up. What we need to do now is provide some identification information for the users. After all, they'll want to know a little something about the program they're installing. The following procedure will show you how to provide that information.

Try This

1. Click the Resources tab and you'll see the Resources – String Table\English window shown in Figure 22-21. If you had requested multiple languages, then there would be one string table for each language. InstallShield normally opens the first language in the list by default. Most of the changes we'll make in the Resources – String Table\English dialog box won't affect the program at all. We're making them for identification purposes or to help the user in some way. Changing one of the resource values is much the same as changing a property in one of the other dialog boxes.

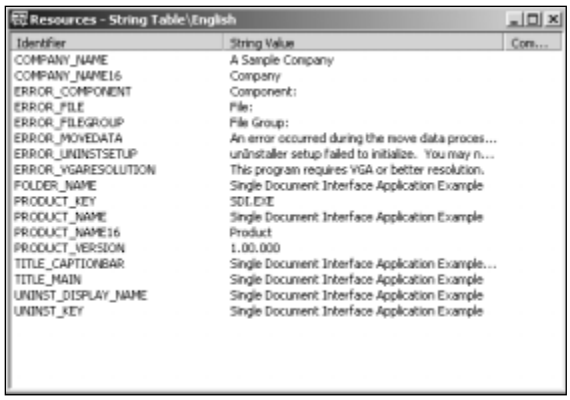

Figure 22-21: InstallShield uses resources about the same way as Visual C++ does.

2. Double-click the Identifier you want to change, and InstallShield will display a dialog box that allows you to make the required change. Figure 22-22 is an example of the COMPANY_NAME identifier. Many of the string table entries already contain values that we defined when creating the installation program using the wizard.
3. Type a new value, if necessary, and then click OK to make the change permanent. You'll see the change in the Resources – String Table\English window.

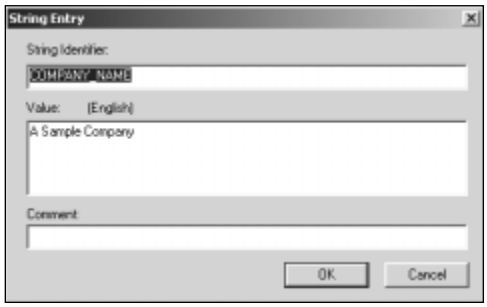

Figure 22-22: Changing properties in the string table is relatively easy.

This three-step process is all you need to define the string table resources we need for the example. Table 22-2 shows the changes I made for the example program — obviously, many of the strings will be different for your installation program.

Table 22-2
String Values for the SDI Installation Program

Identifier	*Value*
COMPANY_NAME	A Sample Company
PRODUCT_KEY	SDI.EXE
PRODUCT_NAME	Single Document Interface Application Example
UNINST_DISPLAY_NAME	Single Document Interface Application Example
UNINST_KEY	SDI.EXE

Determining a Media Type

We've come to the very last task that you must perform from a configuration perspective — simply tell InstallShield what type of media you want to use for distributing your application. In most cases, you'll use CD as the main distribution method and may decide to use floppies as a secondary method if the application is small enough. If your only distribution method is going to be CD, you're done. Otherwise, you need to use the following procedure to create a new media type.

1. Choose the Media tab. You'll see a Media dialog box and window like the ones shown in Figure 22-23. As you can see, the CD media is already in place (as shown by the Default folder in Figure 22-23). You need to use the Media Build Wizard to add another media to your installation program.

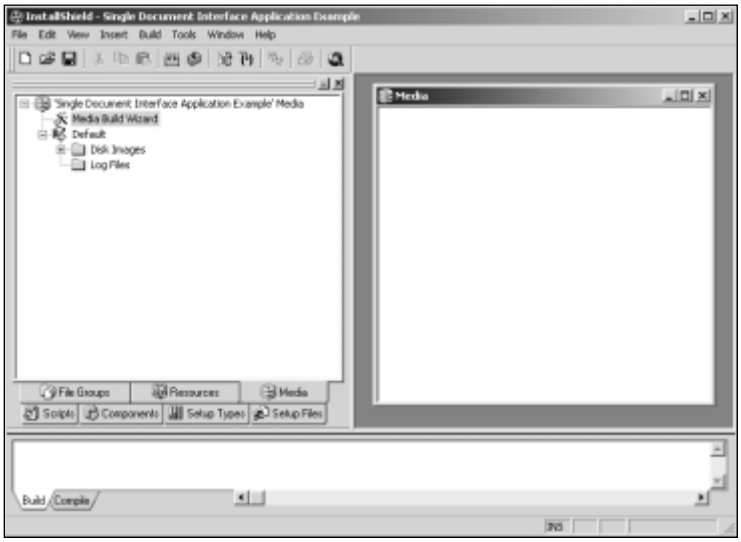

Figure 22-23: The Media tab allows you to define something other than CD as your distribution media.

2. Click the Media Build Wizard entry and you'll see the Media Build Wizard – Media Name dialog box shown in Figure 22-24.

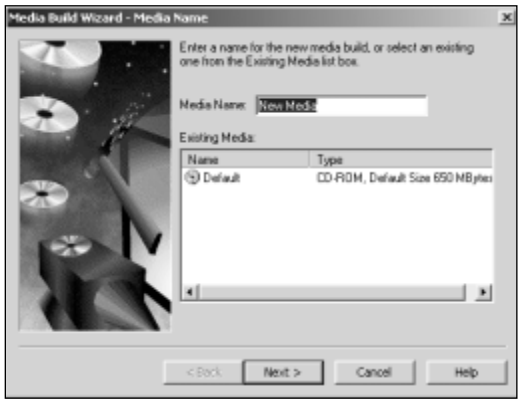

Figure 22-24: You need to define a unique name for each type of media you want to support.

3. Type **My Media**, and then click Next. You'll see the Media Build Wizard – Disk Type dialog box shown in Figure 22-25. Notice that InstallShield supports a wide variety of media types, including 2.88MB floppies.

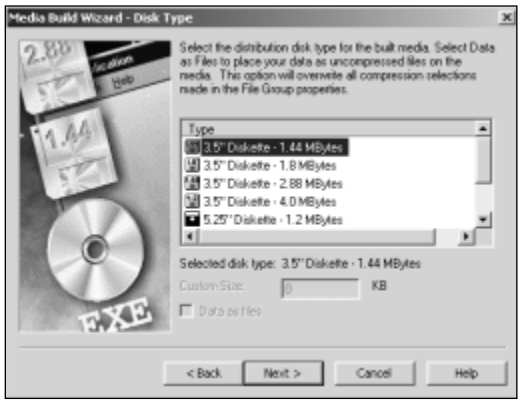
Media Build Wizard - Disk Type
Select the distribution disk type for the built media. Select Data as Files to place your data as uncompressed files on the media. This option will overwrite all compression selections made in the File Group properties.
Type
3.5" Diskette - 1.44 MBytes
3.5" Diskette - 1.8 MBytes
3.5" Diskette - 2.88 MBytes
3.5" Diskette - 4.0 MBytes
5.25" Diskette - 1.2 MBytes
Selected disk type: 3.5" Diskette - 1.44 MBytes
Custom Size:
KB
Data as Files
< Back
Next >
Cancel
Help

Figure 22-25: InstallShield supports a wide array of storage device types.

4. Highlight the 3.5" Diskette – 1.44MBytes option, and then click Next. You'll see the Media Build Wizard – Build Type dialog box shown in Figure 22-26. A full build will compress all the files for your application, create the required CAB files, and create the full-fledged installation program. The Quick Build option is designed for testing purposes. It allows you to see whether your installation program works as anticipated.

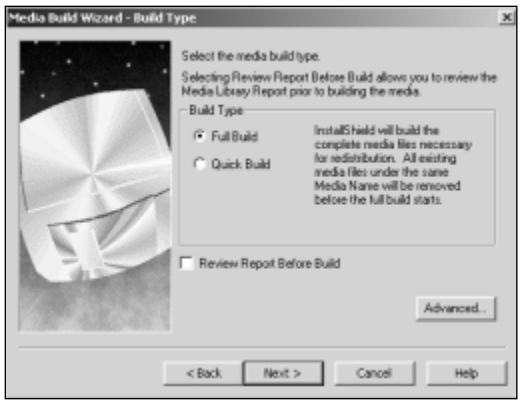
Media Build Wizard - Build Type
Select the media build type.
Selecting Review Report Before Build allows you to review the Media Library Report prior to building the media.
Build Type
Full Build
Quick Build
InstallShield will build the complete media files necessary for redistribution. All existing media files under the same Media Name will be removed before the full build starts.
Review Report Before Build
Advanced...
< Back
Next >
Cancel
Help

Figure 22-26: In most cases, you want a full build for release versions of your installation program.

5. Choose the Full Build option and then click Next. You'll see the Media Build Wizard – Tag File dialog box shown in Figure 22-27. All this dialog box does is allow you to enter your company name and associated application information.

Figure 22-27: You add your company name and application information in this dialog box.

6. Type all the required information into the dialog box. The example program uses **A Sample Company** for the Company Name field, **Single Document Interface Application Example** for the Application Name field, **System Tool** for the Product Category field, and **Simple Utility Program** for the Misc. field.

7. Click Next. You'll see a Media Build Wizard – Platforms dialog box like the one shown in Figure 22-28.

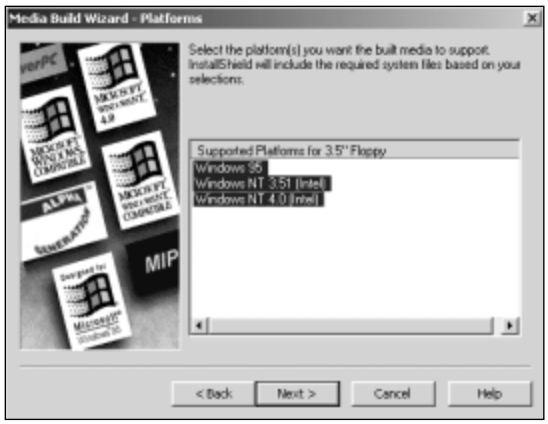
Media Build Wizard - Platforms
Select the platform(s) you want the built media to support. InstallShield will include the required system files based on your selections.
Supported Platforms for 3.5" Floppy
Windows 95
Windows NT 3.51 (Intel)
Windows NT 4.0 (Intel)
< Back
Next >
Cancel
Help

Figure 22-28: This dialog box allows you to choose which platforms you intend to support with a specific media.

8. Click Next. You'll see a Media Build Wizard Summary dialog box like the one shown in Figure 22-29. This is your last chance to verify the settings you've used.

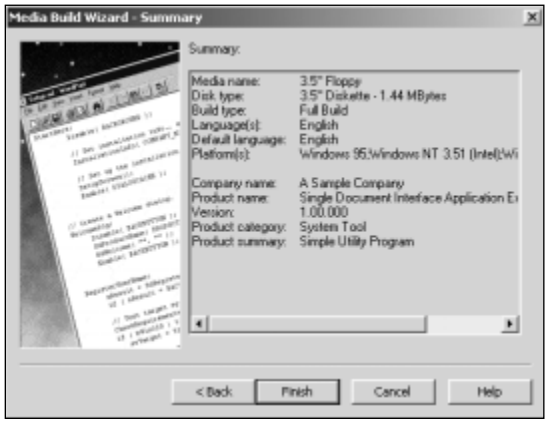

Figure 22-29: Use this dialog box to ensure your build settings are correct.

9. Verify your settings and then click Finish. InstallShield will create the new build that you requested.

The Visual C++ 6 Edition of InstallShield may present an error message, at this point, saying that it doesn't have enough memory to create the requested build. Ignore the problem and try setting up the build again. If you still can't get it to work, complete the rest of the chapter using the default build setting. What you'll need to do is right-click the Default folder and choose Media Build Wizard . . . from the context menu. Choose the Default option on the first screen of the Media Build Wizard dialog box, and then click Next until you get to the final screen where you got the error before. Click Finish and InstallShield will create the media for you.

At this point, you'll see a Building Media . . . dialog. InstallShield is actually creating your installation program for you. When the process is complete, your Building Media dialog should look like the one shown in Figure 22-30. Just click Finish and you're finished!

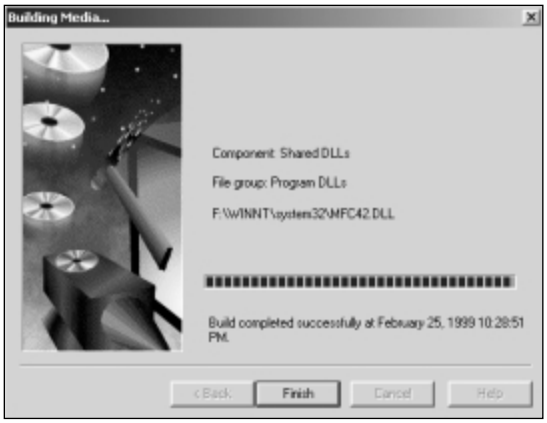

Figure 22-30: You'll see this dialog box once your application installation package is ready for use.

Adding Support for Application Advertising

Application advertising is appearing in more and more installation programs as a way to promote vendor goals. More often than not, application advertising is included in the shrink-wrap model of software because the application vendor normally has the resources to provide a "slide show" presentation. However, there's nothing to stop programmers who create applications for the other packaging models from using application advertising as well.

Of course, the first question is, what does application advertising consists of? There are several major forms of application advertising, each of which is presented in a different way and for a different purpose. The following list provides some ideas of the ways that application advertising can be used to achieve specific goals.

Product Advertising	Most software vendors make more than one product, so it makes sense that to use the installation screens from one program to advertise the benefits of getting another product from the same vendor. However, you don't have to use this method to advertise freestanding products. A graphics application that I recently installed used this type of advertising display to sell add-on products and graphics libraries that they produce. A company could also use this method to advertise special add-ons that are available to specific departments or to advertise other products that work well with the one being installed.
Support Method Notification	Getting help is a major concern for every user, it doesn't really matter which environment you're talking about. Shrink-wrap software tends to advertise the URL for online help, and in some cases, the phone number for the technical support line. Obviously, both the shareware and small-to-medium-sized-business developer could do the same thing. An enterprise programmer could use this kind of advertising to tell the user about the company's help desk or any number of other forms of help that might be available. The bottom line is that you want to use this form of advertising to tell the user where to get help before they get frustrated.
New Feature Listing	Most applications today are so feature rich that it's hard to know what new features they really do support. Looking at the "What's New" section of the help file or README isn't much help for many readers, especially if the list of new features is relatively long. Advertising the most important new features during the installation process is one way to make sure that the user gets the maximum benefit from your application almost immediately.
What's Next Instructions	There are special situations when an application package may contain offerings from more than one vendor, each of which requires it's own installation. This is just

	one situation where you can use an advertising screen to tell the user what to do next — especially if there are several paths that the user could follow, including doing nothing at all.
Product Registration	Shrink-wrap packages almost always contain at least one advertising screen that tells about the benefits of registering the product. There are limited situations where the other packaging models could benefit from this type of advertising as well. For example, if you're an enterprise programmer who has set the program up to install automatically, a registration feature could be a check to ensure the product installed properly. The machines of users who don't register could be checked for potential installation problems.

Now that you have some idea of why to include some advertising dialog boxes, let's look at what you need to do to implement them. InstallShield doesn't provide any automated method for adding nonstandard dialog boxes — you have to modify the script to do it. Every InstallShield installation program comes with a single default script file named SETUP.RUL. This is the file that you need to modify to change the script for your installation program.

There are three steps you need to perform to add specialized dialog boxes to your installation program:

1. Add two statements to make the Sd dialog boxes available within the SETUP.RUL script: #include "sddialog.h" and #include "sddialog.rul". If you use the Project Wizard, this step is performed for you automatically.
2. Create a function to display the special dialog box. InstallShield includes a number of predefined dialog boxes that you can choose from. If you don't find a dialog box that you like, you'll have to use Visual C++ to create a resource DLL, then display the contents of that DLL within InstallShield using the scripting language. This is one place where you'll find that the limitations of the Visual C++ 6 Edition can really cause problems because most of the special display dialog boxes are missing.
3. Modify the ShowDialogs( ) function so that it calls your special dialog box display function. All you need to do is place your specialty function in the list of functions already contained in the ShowDialogs( ) function.

Summary

The one thing this chapter should have done is given you a better appreciation of application packaging. Making sure the user gets a great first impression of the application you've spent so much time creating is essential. In fact, you should always think of packaging elements such as your installation program as your ambassador to the user. The impression that the user gets when installing your application is long-lasting and could affect how they view your application during actual use. Here are some additional conclusions that can be drawn from the contents of this chapter:

- It's important to choose the right packaging model, or a combination of packaging models when you're working with an application that fits within more than one environment. You can easily divide the packaging environments that most developers will work in into small to medium-sized business, enterprise, shareware, and shrink-wrap.
- There are a lot of application environmental factors to consider when building your application package, such as the application size, complexity, and modularity, along with the type of audience that the package is designed to accommodate.
- Creating a list of the files that you need to include with your application is important. The three most useful utilities for creating this list are QuickView, DumpBin, and Depends. Of the three, Depends provides the most complete information, and DumpBin is useful for automation that relies on text files.
- Using InstallShield to create your application installation program is normally a five-step process: create the application shell, define the various components, set up file groups, create any required resources, and determine which media type you want to use.
- Application advertisements can help just about any developer to help the user understand the product better, even though they're normally associated with shrink-wrap software.

✦ ✦ ✦